Rapport d'alternance en master

Création d'un outil d'administration

Julien Ivars

UHA 4.0.4

Master 1 Informatique et mobilité

Unit Solutions

2021-2022

Tuteur professionnel

M. Cédric MARTIN

Tuteur pédagogique

M. Mounir ELBAZ

Remerciements :

J'aimerais remercier monsieur le directeur d'Unit Solutions M. **Thierry MOEBEL** pour m'avoir donné l'opportunité de rejoindre l'entreprise et d'effectuer ma première année de master en alternance.
Je le remercie également d'avoir pris en compte mes intérêts en me confiant un projet captivant, correspondant parfaitement aux attentes de mon année.
De plus, je suis reconnaissant qu'il ait prolongé mon contrat pour l'année prochaine, me permettant ainsi de poursuivre mes études en master au sein de l'entreprise.

Je souhaite exprimer ma gratitude envers M. **Cédric MARTIN**, mon tuteur en entreprise, pour son accompagnement tout au long de l'année sur le projet. Sa transmission de connaissances techniques et ses explications sur l'architecture et le fonctionnement du projet ont été d'une grande aide pour moi.

Je remercie chaleureusement tous mes collègues chez Unit Solutions pour leur partage de connaissances, leur bonne humeur et leur soutien, malgré les difficultés sanitaires qui ont limité les occasions de partage en personne.

Je tiens à exprimer ma reconnaissance envers toute l'équipe pédagogique de l'UHA 4.0, notamment M. **Mounir ELBAZ**, M. **Pierre-Alain MULLER**, M. **Florent BOURGEOIS**, M. **Daniel DA FONSECA**, M. **Pierre SCHULLER** et Mme. **Audrey BRUNSPERGER,** ainsi que les étudiants de l'UHA 4.0. Leur soutien, leur partage de connaissances, leur accompagnement et leurs conseils tout au long de l'année m'ont permis de mener à bien mon projet professionnel.

Enfin, je souhaite exprimer ma gratitude envers les relecteurs de ce rapport pour leurs précieux conseils, qui m'ont permis de mener à bien l'écriture de ce rapport.

Sommaire :

Introduction

Après trois années d'études à l'UHA qui ont résulté en l'obtention de ma licence professionnelle développeur informatique, j'ai décidé de poursuivre mon parcours en intégrant le cursus master proposé par l'UHA 4.0.

Cette première année en master a été réalisée en alternance au sein d'Unit Solutions, la même entreprise qui me suivait déjà l'année passée dans mon projet d'études. J'ai pu apporter ma contribution au projet InfKuba et les autres applications reliées. L'application InfKuba permet de gérer l'inspection et la maintenance d'ouvrages d'art.

J'approfondirai donc dans ce document les développements que j'ai pu effectuer sur un projet qui m'a été confié durant les différentes périodes passées en entreprise.

Dans ce document, je commencerais par présenter le contexte qui m'a amené à intégrer le master et l'entreprise Unit Solutions.
J'aborderais par la suite l'objectif qui m'a été fixé durant cette année d'alternance et le projet sur lequel j'ai pu contribuer.
Je présenterais ensuite les développements que j'ai pu mener en présentant la solution finale validée par mon tuteur en entreprise.
Enfin je ferais une brève conclusion pour exposer ce que ce temps en entreprise a pu m'apporter ainsi qu'à l'entreprise.

Certains termes utilisés dans ces documents sont en allemand pour éviter de dénaturer le contexte. J'ajouterais une note de bas de page pour apporter une traduction aux mots concernés si nécessaires.

Recontextualisation

Dans cette première partie, je vais recontextualiser le concept de la formation, présenter mon parcours scolaire et je présenterais également l'entreprise qui me fait confiance pour mon alternance.

1. Présentation de la formation

L'UHA 4.0 propose pour ses différentes formations des formats légèrement différents que des cursus plus traditionnels, pour permettre une immersion dans le monde professionnel tout en poursuivant une formation universitaire.

Durant la formation, les étudiants doivent réaliser un stage d'une durée minimale de 6 mois en complément de la période de formation de même durée. Cette immersion en mode projets et stages en entreprise, fournit aux étudiants les outils les plus importants pour se lancer dans le monde professionnel sans problèmes.

Durant la première année, l'introduction au développement permet de fournir les outils élémentaires pour apprendre la logique de la programmation.

La seconde année permet d'apporter une nouvelle brique à cet apprentissage en y abordant le concept de programmation orientée objet avec des langages tels que Java ou C#.

La troisième et dernière année du parcours de Licence permet d'apporter en complément du développement, une méthodologie approfondie sur le module de gestion de projet en mode agile.

Après ces trois années, il est possible, sur admission, de rejoindre le parcours master au sein de l'UHA 4.0. Ce parcours master permet d'aborder les fondements de la recherche. Il y aborde également des thématiques un peu plus variées telles que la cybersécurité, l'algorithmie…

Sous forme d'alternance également, le master impose aux étudiants une période en entreprise un peu plus soutenue avec non pas 6, mais 9 mois en entreprise et 3 mois en cours. Les étudiants doivent néanmoins réaliser en parallèle de leur projet en entreprise, les différents projets donnés par les encadrants à l'UHA 4.0.

2. Mon parcours

Après l'obtention de mon baccalauréat scientifique, j'ai pu intégrer l'UHA 4.0 et suivre mes trois premières années d'études au sein de l'école. Ces trois années de formation m'ont permis de réaliser deux premiers stages de 6 mois dans deux entreprises différentes et ma troisième année en alternance chez Unit Solutions AG.

L'obtention de mon diplôme de Licence a conclu mes trois premières années d'études supérieures. J'ai par la suite décidé de poursuivre en rejoignant le cursus Master, toujours à l'UHA 4.0, pour compléter mes acquis et acquérir de nouvelles compétences. Cette formation en alternance met la priorité sur le temps en entreprise en le faisant passer de 6 mois à 9 mois.

Le format du cursus en master reste identique au format de la troisième année de Licence. Les premiers mois de formations requièrent un investissement accru afin de mener à bien les différents projets apportés par les formateurs tout en menant le projet de l'entreprise d'accueil en parallèle.

J'ai eu l'occasion de poursuivre mon alternance au sein de la même entreprise qui m'avait déjà accordé sa confiance lors de ma dernière année de licence. J'ai continué l'apprentissage des technologies employées dans l'entreprise ce qui m'a permis de progresser notamment en C# que je ne connaissais particulièrement avant mon alternance et le framework[1] Angular que je n'utilisais que de manière très sommaire.

Travailler sur un projet avec des développeurs expérimentés m'a permis d'acquérir de nombreuses connaissances, autant sur la gestion de projet pour travailler avec efficacité, que dans le développement en gardant des projets organisés et clairement architecturés maintenir les projets dans le temps.

[1] Un framework est un ensemble de composants logiciels structurels permettant d'aider à la création d'un logiciel.

3. L'entreprise

Pour cette première année de master, l'entreprise **Unit Solutions** a donc décidé de poursuivre mon contrat pour me permettre de mener à bien mon projet d'études.

Unit Solutions est une entreprise suisse, basée à Allschwil dans le canton de Bâle-Campagne. Fondée en 1986 premièrement sous le nom CADRZ, elle est aujourd'hui dirigée par M. **Thierry MOEBEL**. Elle est premièrement dédiée à la création d'un cadastre numérique pour la ville d'Allschwil.

La philosophie de l'entreprise va par la suite changer pour finalement réaliser ses propres logiciels et en faire la maintenance. L'entreprise compte actuellement une vingtaine d'employés pour le développement des différentes solutions, le support et l'administratif.

Les développements au sein de l'entreprise reposent sur quatre gros projets :

- Langsam Verkehr[2] est une solution visant la gestion des sentiers de mobilité douce en Suisse. Cela comprend la gestion des sentiers de randonnée ou les pistes cyclables.
- Kuba et InfKuba sont les deux solutions principales portées par Unit Solutions. Ces deux logiciels sont relativement similaires dans leur but, mais leur conception est totalement différente. Kuba est sous forme de client lourd (application à installer) tandis qu'InfKuba est une application web qui ne nécessite rien d'autre qu'un navigateur. Elle est la version moderne de Kuba.
- Observo est un projet servant d'extension au projet InfKuba, ou alors d'outil totalement indépendant pour gérer des domaines d'applications qui lui sont propres. Il permet par exemple de gérer de petits ouvrages, du mobilier urbain…

Durant mon alternance, j'ai pu intégrer l'équipe chargée des développements pour la suite logicielle InfSuite. Entouré de plusieurs collègues, j'ai pu développer mes bases de connaissances sur le projet et les différentes technologies utilisées.

[2] Traduction : Mobilité Douce

EXISTANT ET CAHIER DES CHARGES

Dans cette partie, je vais en premier lieu présenter l'application InfKuba ainsi que son architecture. J'apporterais par la suite des informations sur le besoin que génère une application d'une telle envergure. Enfin, j'expliquerais comment a été imaginée la conception de la solution développée.

1. Présentation d'InfKuba

Début **2016**, les différents cantons suisses étaient sur le client lourd Kuba. Ce logiciel était payé par L'OFROU[3] pour une meilleure gestion des infrastructures routière en Suisse.

En **2019,** elle lance un appel d'offres dans le but de réduire les coûts liés aux applications qu'elle utilise. L'application Kuba étant une application hautement complète et complexe, le budget n'était alors pas forcément adapté à tous les cas d'usage des différents cantons, certains ayant des besoins différents en termes de gestion.

Unit Solutions décide alors de se lancer dans la conception d'une nouvelle solution nommée InfSuite, qui reprend le principe de l'application Kuba. La grande différence sur cette nouvelle application, est le découpage plus fin des différentes fonctionnalités. Elle permet de proposer aux différents clients de ne leur octroyer l'accès à certaines fonctionnalités que s'ils en ont réellement besoin et permet ainsi d'affiner le budget à leurs besoins.

Les principaux objectifs de l'entreprise étaient donc :

- De n'implémenter que les fonctionnalités que les cantons seraient susceptibles d'utiliser.
- Vendre l'application à l'OFROU appuyé par la nouvelle philosophie de l'application qui réduit les coûts de l'application.
- Livrer une version utilisable au bout d'une année et demi maximum pour permettre aux cantons de tester l'application et de s'adapter au nouveau système.

Adoptée par la suite par la majorité des cantons suisses, l'application InfKuba est une application géographique basée sur des technologies web récentes. Le

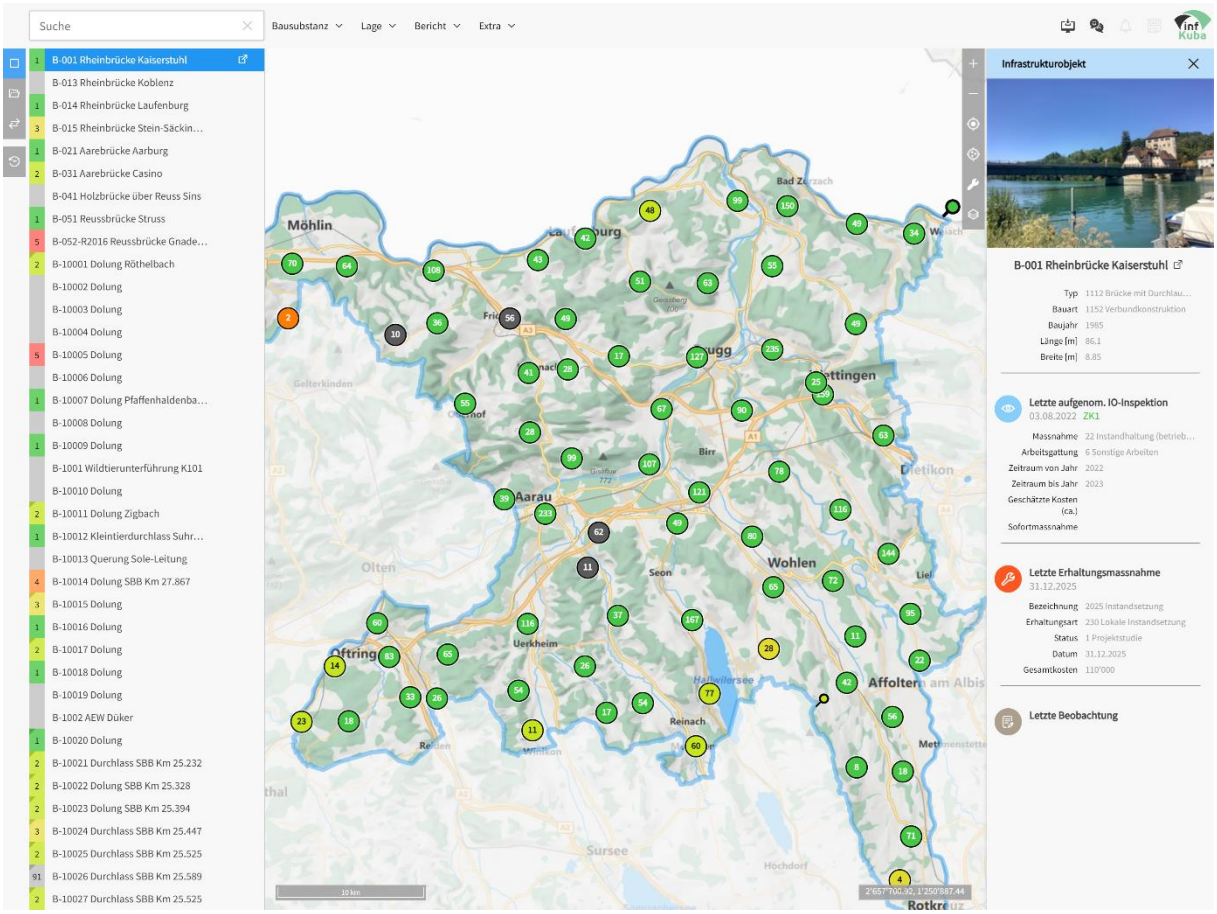
[3] L'**O**ffice **F**édéral des **ROU**tes est l'autorité suisse compétente pour l'infrastructure routière.

concept de l'application est de pouvoir faire l'état et le suivi d'ouvrages d'art dans le temps. Par exemple, un organisme responsable de la préservation des ouvrages d'art vient faire des constatations à une certaine date de l'ouvrage et enregistre toutes les données relatives dans l'application.

Lorsqu'une nouvelle expertise sera programmée sur cet ouvrage, de nouvelles observations seront ajoutées et avec ces nouvelles données l'application fournira un résumé de l'évolution de cet ouvrage. Elle fournira des informations sur la détérioration de l'ouvrage, s'il y a un comportement particulier, s'il faut planifier une intervention de restauration, etc.

Toutes ces données sont accessibles depuis n'importe où et depuis n'importe quel terminal tant que le client à accès à une connexion internet et un compte ayant les droits requis pour accéder aux données. L'application appartient à la suite logicielle « InfSuite ». Cette suite logicielle comprend plusieurs autres outils basés sur le même fonctionnement et sur la même application qu'InfKuba. Ces outils permettent d'effectuer des constatations plus spécifiques. Par exemple, sur l'outil InfAqua, il est possible de faire l'observation de cours d'eau (lit de rivières, berges…), InfRail des voies ferrées, InfVias, de routes… Nous garderons l'outil **InfKuba** comme sujet principal pour l'écriture de ce document, mais le fonctionnement des différents outils ne varie que très peu.

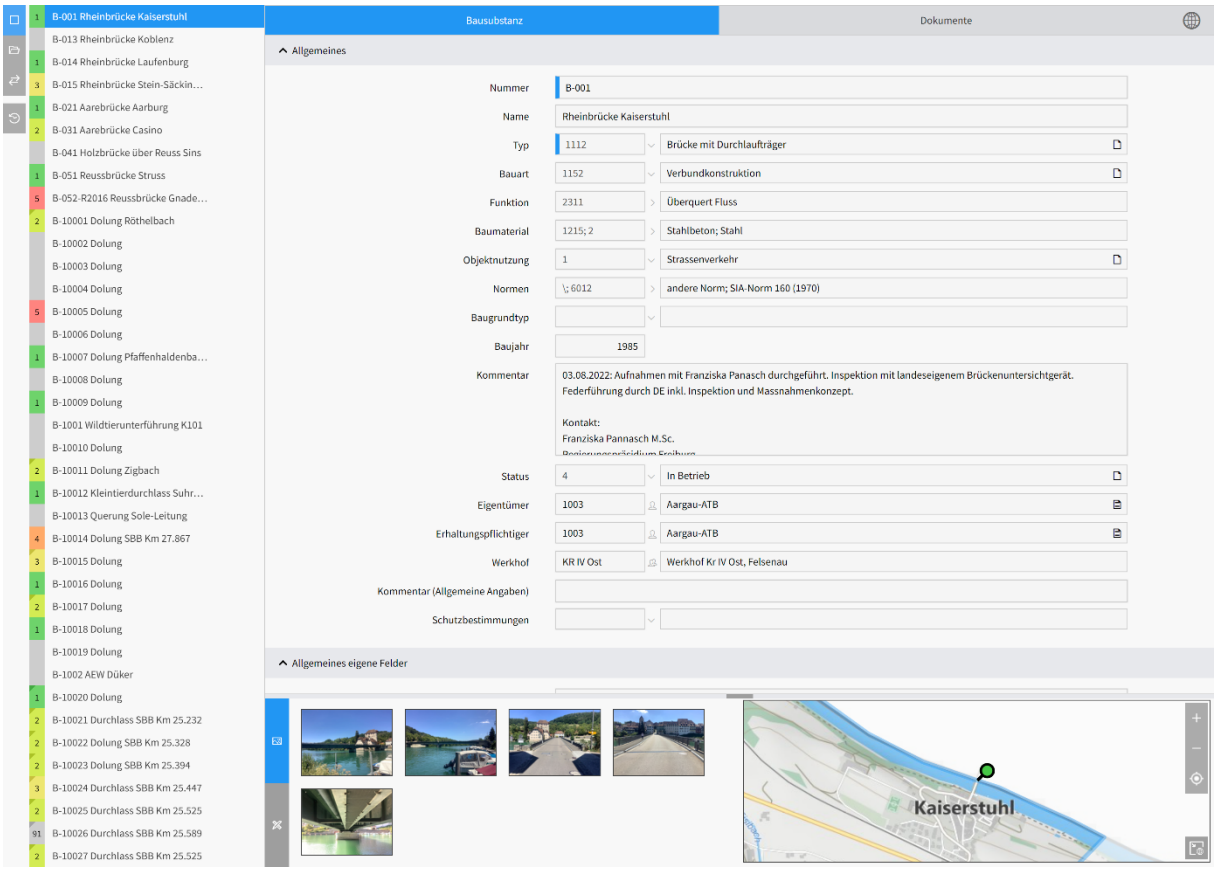
L'application présente de manière simplifiée les données relatives aux différentes constatations sur une carte. Elle représente sous forme de pastilles colorées la note moyenne des ouvrages par zones géographiques lorsque plusieurs ouvrages se trouvent très proches les uns des autres comme le montre l'image ci-dessous.



A - Aperçu de l'application InfKuba

En fonction du niveau de zoom, les différents ouvrages se séparent les uns des autres et permettent de voir la dernière note calculée pour l'ouvrage sous forme d'une petite épingle qui représente l'objet d'infrastructure. La couleur varie du vert au rouge indiquant respectivement que l'ouvrage est en bon état ou qu'il faut intervenir le plus rapidement possible.

En ouvrant les détails de l'objet on peut retrouver un onglet regroupant les informations relatives à l'ouvrage d'art. On y retrouve des données basiques comme son nom, le canton propriétaire de l'ouvrage, ses dimensions, des commentaires, sa position géographique, des photos, etc.



B - Aperçu des détails de l'ouvrage dans InfKuba

D'autres onglets permettent de retrouver des informations plus précises telles que les observations de l'ouvrage, des graphiques sur l'évolution de l'ouvrage dans le temps, visualiser l'ouvrage en 3D (si fourni par le client) …

Chaque ouvrage appartient à un canton, mais il peut être prêté à d'autres cantons, comme dans le cas où un pont serait partagé entre deux cantons par exemple. Un autre exemple d'ouvrage partagé est lorsque la gestion est déléguée à une ville plutôt qu'au canton. À ce moment, dans l'application, l'ouvrage appartient au canton et est « prêté » à la ville qui en est chargé.

Chaque donnée générée à partir de l'application, est rattachée au client qui en est à l'origine, ce qui permet d'avoir un historique en cas de problème.

Pour permettre de rendre l'application plus légère, l'application est basée sur une architecture trois tiers.

Le premier tiers de l'application correspond à la base de données. Cette base de données est une base PostgreSql, particulièrement adaptée pour traiter les données géographiques, mais est également réputée pour sa rapidité comparée

à une base de données type MySql. C'est ici que toutes les données sont stockées ainsi que les relations entre chaque donnée existante.

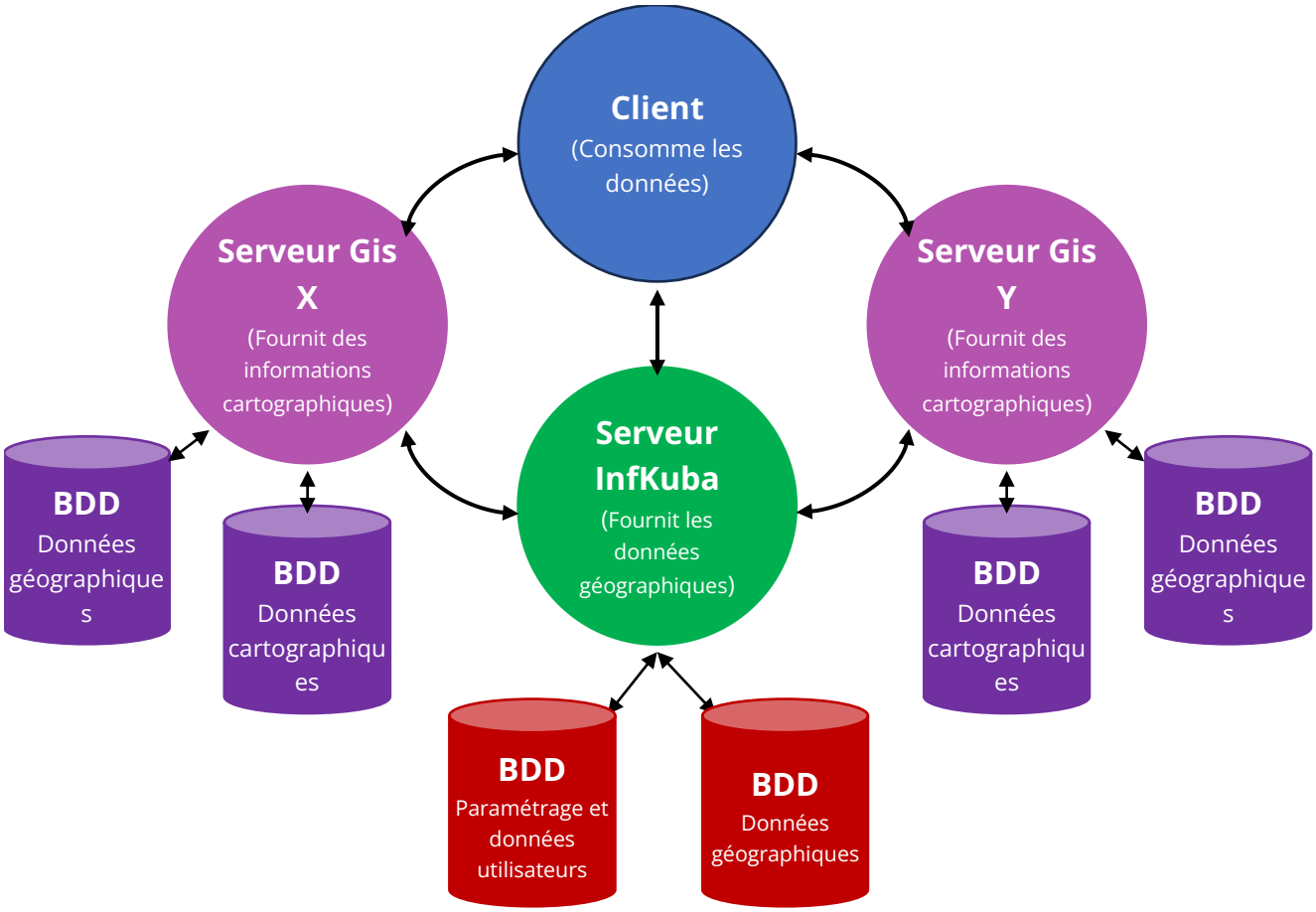
Le second tiers de l'application correspond à l'API. Une API web, est une interface de programmation qui permet à des applications informatiques de communiquer entre elles via Internet. Elle définit les règles et les formats de données pour faciliter l'échange d'informations entre les différentes applications.

Cette API est le serveur back-end qui permet de faire la relation entre le monde extérieur et les données brutes stockées en base de données. Le serveur est développé en C# avec l'ORM Entity Framework. Cette interface permet notamment de mettre en forme les données pour qu'elles correspondent aux besoins du troisième et dernier tiers.

Le dernier tiers correspond à la partie visible de l'application. Appelé frontend, il met à disposition de manière visuelle et simplifiée les données. Ce dernier tiers permet également à l'utilisateur de créer, modifier ou effacer des données. Il est développé en TypeScript avec le framework Angular pour permettre de créer une application dynamique et réactive.

Le dernier service complémentaire est un serveur GIS. Ce serveur permet de fournir des informations cartographiques comme des adresses. Il permet également de délivrer les fonds de cartes. Ils peuvent être hébergés et entretenus par Unit Solutions, ou alors par d'autres entreprises comme le fond de carte Open Street Map appartenant à la fondation du même nom.

Le schéma ci-dessous représente les trois tiers de l'application communiquant ensemble et le serveur GIS permettant de fournir des informations complémentaires.



C - Représentation de l'architecture d'InfSuite et de ses dépendances

L'application propose à l'utilisateur de personnaliser toute l'application. Cela comprend par exemple quel fond de carte afficher, quelles données afficher, personnaliser l'affichage de ces données tel que le type d'affichage des épingles… Ces configurations sont propres à l'utilisateur et sont sauvegardées en base de données, en parallèle de toutes les autres données de l'application.

2. Présentation d'InfAdmin

Pour paramétrer l'application InfKuba, un outil d'administration a été pensé pour manipuler les données et les configurations de l'application. Cet outil, apparu au même moment qu'InfKuba, a pour but premier de gérer les comptes des différents utilisateurs et leur octroyer les droits nécessaires.

Elle reprend la même architecture que l'application mère InfKuba pour simplifier les développements, mais ne partage pas de code avec elle. Cependant, les bases de données sont partagées. Le but de l'application est de modifier la configuration qui est stockée en base de données, il est donc possible de cloisonner les services requis dans un environnement à part.

Pour la partie cliente, hormis les services permettant de communiquer avec la partie backend, les implémentations de l'interface utilisateur sont relativement similaires à InfKuba. Cela permet de garder une certaine harmonie entre les deux applications malgré leur cloisonnement. On y retrouve le même fonctionnement avec des modes qui s'axent chacun sur un type de configuration différent.

Le premier mode, cité précédemment, est le mode utilisateur. Dans ce mode, on retrouve l'ensemble des comptes utilisateurs enregistrés. On peut, pour chaque compte, lui ajouter un mandant, lui octroyer les droits de modifier les données de ce dernier, de le limiter à un accès lecture seulement, etc.

Le second mode, présent antérieurement à mes développements sur l'outil est le mode « gestion des IPads ». Une application mobile d'InfKuba est développée pour permettre d'accéder à l'application et de saisir des données même hors ligne.

Pour gérer les appareils qui ont accès à l'application, il faut les enregistrer et octroyer les droits à un utilisateur sur l'appareil, c'est dans cet onglet que tout est configurable.

Enfin, le troisième et dernier onglet développé avant mon arrivée sur le projet est l'onglet « produit externe » qui permet de gérer l'accès des autres produits de Unit Solutions aux ressources d'InfKuba.

L'outil est développé en parallèle d'InfKuba pour les besoins qui requiert des modifications très bas niveau dans l'application. Comme cité précédemment, c'est ici qu'on y gère les utilisateurs, la configuration des utilisateurs et comme nous le verrons plus loin dans ce document, de nombreuses autres fonctionnalités faisant appel aux fonctionnalités centrales d'InfKuba.

3. Etat de l'art et problématique

Sur le plan technique, toute l'application repose sur les « data-owner ». Un « data-owner » ou propriétaire de données en français est l'entité qui répertorie toutes les informations primordiales telles que le type de produit (InfKuba, InfVias…), la monnaie utilisée (CHF, €…) et de nombreuses autres configurations.

Un mandant correspond à un client. Par exemple, le canton de Bâle-Campagne est un client et possède son mandant dans l'application. Pareil pour l'Eurométropole de Strasbourg qui est cliente et qui possède son propre mandant. Il peut y avoir plusieurs mandants rattachés à un data-owner.

À mon arrivée sur le projet, la majorité des développements effectués sur InfAdmin concernaient, comme cité précédemment, la création de compte pour les utilisateurs de l'application, la gestion des appareils mobiles et la gestion des permissions d'accès aux données depuis d'autres applications.

Toutes les autres configurations d'InfKuba, que ce soit concernant la création et la gestion de mandant, data-owner, champs propres, cartes, etc. passaient par l'utilisation de commandes SQL. Un budget était alloué pour la création de ces données manuellement par le billet de requêtes SQL écrites à la main.

La création des données manuellement permet de raccourcir le temps initial pour créer ces dernières. Il « suffit » d'écrire la requête une première fois, tester qu'elle fonctionne comme attendue puis modifier certaines valeurs en fonction des besoins.

Pour permettre aux clients d'accéder à l'application, il faut donc initialiser ces deux entités mère : un data-owner et au moins un mandant. Pour créer un data-owner on utilise par exemple la requête SQL ci-dessous.

```sql
INSERT INTO public.add_dataowner_mkb (dao_id, dao_dab_id, dao_intern_nm,
dao_extern_nm, dao_tx, dao_create_dt, dao_createuser_vl, dao_change_dt,
dao_changeuser_vl, dao_simple_io_vl, dao_simple_onsp_vl, dao_sustaining_vl,
dao_onsp_autostate_vl, dao_graph_state_vl, dao_io_autoerma_vl,
dao_onsp_autost_ex_vl, dao_beobachtung_vl, dao_type_vl, dao_product_vl,
dao_map_center_tx, dao_map_max_extent_tx, dao_over_ch_nat_vl,
dao_over_ch_ramp_vl, dao_show_roadworks_vl, dao_enabled_modus_cd)
VALUES ('XXX', 'XXX', 'XXX', 'XXX', NULL, now(), 'XXX', now(), 'XXX', false,
false, false, false, 0, false, false, false, 1, 1, NULL, NULL, NULL, NULL,
NULL, 1);
```

D - Exemple de requête SQL pour créer un DataOwner en BDD

Le data-owner n'est pas la seule entité qu'il faut initialiser. Il faut également créer un mandant, octroyer les permissions à l'utilisateur, ajouter des

configurations supplémentaires essentielles à l'application pour le data-owner… Toutes ces entités, à créer en base de données, multiplient le nombre de requêtes qu'il faut exécuter pour atteindre la configuration minimale pour permettre à InfKuba de fonctionner.

L'utilisation de cette méthode reste relativement peu efficace pour plusieurs raisons. Avec l'implémentation progressive de nouvelles fonctionnalités dans InfKuba, les données de configuration en base et les liaisons de tables se sont démultipliées.

Cela implique donc de modifier les requêtes écrites précédemment, mais aussi d'y implémenter les nouvelles configurations et les nouvelles relations. De plus, une erreur lors de l'écriture de ces requêtes peut être plus difficile à trouver lors du debug.

Un problème bloquant est le nom des colonnes. Soit l'utilisateur connaît les structures des bases de données par cœur et peut remplir la requête lui-même sans aucune aide, soit il suit une documentation contenant les différentes colonnes expliquant ce à quoi chacune d'elles correspond et doit remplir colonne par colonne les informations.

Un autre problème qui se pose concernant cette méthode sont les compétences requises. Si un développeur ayant les connaissances nécessaires n'est pas présent pour s'occuper de faire l'insertion en base de ces données et qu'aucune autre personne n'est qualifiée pour le faire, le client risque de devoir attendre plus longtemps que prévu pour avoir accès à ses données.

Enfin, en dernier point bloquant, l'utilisation de ce genre de script est limitée d'accès aux personnes internes à l'entreprise. Les clients n'ayant aucun droit d'accès direct aux bases de données par mesure de sécurité, il ne leur est donc pas possible de manipuler leurs données eux-même.

Avec les différents points bloquants que j'ai pu exposer ci-dessus, je peux ainsi me demander : **Comment administrer un nombre conséquent de données dans une application de grande envergure ?**

En partant de cette problématique et des différentes informations que j'ai en ma possession, je peux dresser un cahier des charges pour essayer d'y trouver une réponse.

4. Le cahier des charges

Le but est donc de trouver une alternative aux scripts SQL qui doit cependant respecter un certain nombre de critères.

En premier, la solution développée doit au minimum reprendre les mêmes fonctionnalités que ce qu'il était possible de faire au travers de l'exécution manuelle de requêtes SQL.

Elle devra également être suffisamment flexible pour pouvoir y implémenter de nouvelles fonctionnalités. Comme vu précédemment, InfKuba évolue, il faut donc permettre d'implémenter les nouvelles fonctionnalités sans devoir passer par de trop nombreuses modifications de l'outil, sans quoi l'argument de productivité serait caduc.

Un point important est que l'outil doit être simple d'utilisation. Il a pour but de remplacer une partie complexe de la création de configuration, il faut donc que la solution tienne sa promesse et permette à n'importe quel utilisateur même non expérimenté d'utiliser l'outil.

Enfin, il doit fournir un environnement sécurisé. L'application contient des données qui appartiennent aux différentes entreprises/institutions et il est donc primordial de contrôler l'accès à l'application pour ne pas permettre à n'importe qui de modifier les données ou de s'octroyer les droits là où il le désire.

Solution et réalisation

Dans cette dernière partie je vais présenter les différentes pistes de réflexions pour la réalisation d'un nouvel outil qui correspond au cahier des charges imposé, puis je présenterais l'aspect technique des différents développements que j'ai pu effectuer et enfin je présenterais mes contributions en dehors d'InfAdmin.

1. Recherche d'une solution

Les décisions d'améliorer l'application sont majoritairement prises par le chef de projet s'il voit que le besoin est trop important. Ce besoin peut aussi être exprimé par le patron de Unit Solutions qui essaye de garder un œil attentif sur toute faiblesse potentielle dans les différents projets.

Il a été constaté que le problème concernant l'exécution de scripts SQL lors de la création et de la configuration initiale des objets au sein de l'application InfKuba devenait un peu trop récurrent et qu'il fallait régir.

J'ai ainsi été chargé de trouver une solution pour remplacer la méthode manuelle par une méthode un peu plus automatisée. Plusieurs solutions potentielles se sont présentées.

Lors de ma troisième année de licence, j'avais déjà eu la charge de réaliser un outil de conversion pour l'application. Cette application est développée en C# et est dite CLI[4].

Je me suis donc orienté sur ce précédent projet pour établir une piste de réflexion sur la création d'un outil de configuration en tant qu'application CLI. Lors des différentes discussions qui ont pu avoir lieu au sujet de cette application, en est ressorti différents points permettant de peser le pour et le contre.

La création d'une application CLI comme celle-ci permet de grandement réduire les temps de développements. En effet, il n'est pas nécessaire de développer d'interface graphique, uniquement de récupérer les paramètres que l'utilisateur fournit en entrée de programme puis l'application peut effectuer les différents traitements.

L'application est légère et son fonctionnement linéaire permet de repérer très rapidement les potentielles erreurs pour les corriger. Cela permet également de

[4] Une application **C**ommand **L**ine **I**nterface est une application qui s'exécute dans la console du système d'exploitation.

simplifier l'implémentation de nouvelles fonctionnalités puisqu'elle ne contient que les éléments strictement nécessaires à l'initialisation des configurations.

Cependant, son utilisation est limitée par de nombreux facteurs. Une première limite est la portabilité. Pour chaque nouvelle version, les fichiers de l'application sont modifiés et il faut qu'ils soient disponibles pour chaque personne voulant l'utiliser à n'importe quel moment.

C'est une application fonctionnant en mode « pipeline », elle prend des paramètres en entrée et applique les différentes fonctions pour initialiser les données nécessaires. Il n'est pas possible de modifier les données, ou alors une implémentation de cette fonctionnalité rendrait l'application peu agréable à utiliser parce que trop complexe.

Enfin, ce genre de solution n'est pas possible par le fait que les bases de données de Unit Solutions ne sont pas accessibles depuis l'extérieur. Tout est sécurisé et cloisonné pour empêcher des failles de sécurité. Ainsi, même si l'utilisateur possède l'application CLI, il lui est impossible de contacter les bases de données depuis l'extérieur et donc l'application ne peut rien faire.

Les pistes de réflexions se sont alors orientées vers une autre implémentation. L'UHA 4.0, orientée vers les technologies du web, nous forme sur la création d'applications web utilisant une architecture trois-tiers. Ma seconde piste était donc de créer une application web pour permettre l'administration des données.

La création d'une telle application comporte de multiples avantages.

Elle permet premièrement de décharger le client de tout traitement lourd en déléguant les grosses tâches au serveur dédié.

Cela permet aussi de sécuriser l'application. En faisant passer le traitement des données par une API, on laisse au client l'accès qu'aux données auxquelles il est autorisé et on contrôle les données d'entrées.

Et enfin, l'implémentation d'une interface cliente permet de simplifier le processus de création des données, mais également rend possible la modification de ces données en les présentant de manière simple et rangée.

Autre avantage de cette solution, il n'y a pas besoin de réinventer la roue et de partir de rien puisqu'il existe déjà un outil qui répond en partie à notre besoin, InfAdmin. L'idée était donc de compléter l'application existante en y implémentant de nouvelles fonctionnalités plus poussées.

Cette application web utilise l'architecture logicielle API Rest. Ce choix se justifie pour plusieurs raisons : nous cherchons à implémenter des fonctionnalités

lourdes en traitement et complexe côté serveur pour délester le client. Le serveur doit également être distant. Enfin, il est important d'homogénéiser les technologies utilisées dans l'entreprise et principalement avec la solution mère, il est donc logique de réutiliser les technologies et philosophie de fonctionnement de celle-ci.

Méthodes de travail

Pour commencer de nouveaux développements, je présente dans un premier temps un concept technique de la fonctionnalité, en exposant les avantages et les inconvénients à mon chef de projet tout comme j'ai pu le faire précédemment.

Nous avons échangé sur les différents points qui pourraient rester trop vagues comme : quelles sont les implémentations nécessaires dans la création de configuration pour les data-owners et les mandants.

Pour rédiger un document technique, nous utilisons Confluence. Confluence est un logiciel de Wiki collaboratif qui permet de rédiger des documents en collaboration et de les conserver au sein d'une seule et même plateforme. Les documentations sont rédigées pour présenter globalement ce qui est attendu et si elle est à destination d'un autre développeur, elle peut contenir des détails supplémentaires sur la partie technique de l'implémentation.

Il se peut qu'après une relecture de l'article confluence, le projet de développement soit finalement mis en suspens parce que jugé non pertinent. Cela peut arriver lorsqu'une tâche plus importante, apportant des modifications sur les concepts concernés par l'article, arrive entre-temps.

Si les développements sont nécessaires, alors un ticket est créé dans l'outil Jira qui permet de faire de la gestion de projet. C'est dans cet outil que toutes les tâches sont planifiées. Chaque tâche comporte un ordre de priorité allant de très haute priorité à très faible priorité. Si la tâche à une priorité haute, elle sera traitée dans les plus brefs délais, il s'agit majoritairement de bugs ou de features urgentes. Pour les tâches de priorité basse, elles correspondent généralement à des ajouts mineurs dans l'application et seront traitées plus tard.

Ici, il a été estimé que la tâche, impliquant des fonctionnalités pouvant améliorer grandement le processus d'intégration de nouveau client dans l'application, était prioritaire et que les développements peuvent être démarrés directement après validation du concept technique.

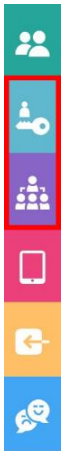
Après la phase de réalisation du concept technique et de la validation de ce dernier, le ticket passe par la phase de réalisation. C'est durant cette étape que les

développements sont réalisés. Je présente donc dans la section suivante l'outil final pour permettre de mieux comprendre la partie technique de l'application.

2. De la théorie à la pratique.

L'objectif des développements est de reprendre la structure actuelle d'InfAdmin et d'y rajouter les nouvelles fonctionnalités. Il faut donc s'adapter à ce qui est déjà réalisé. Certains composants n'ayant pas forcément été conçus dans l'objectif d'être réutilisés par la suite, il faut soit les remplacer, soit les modifier.

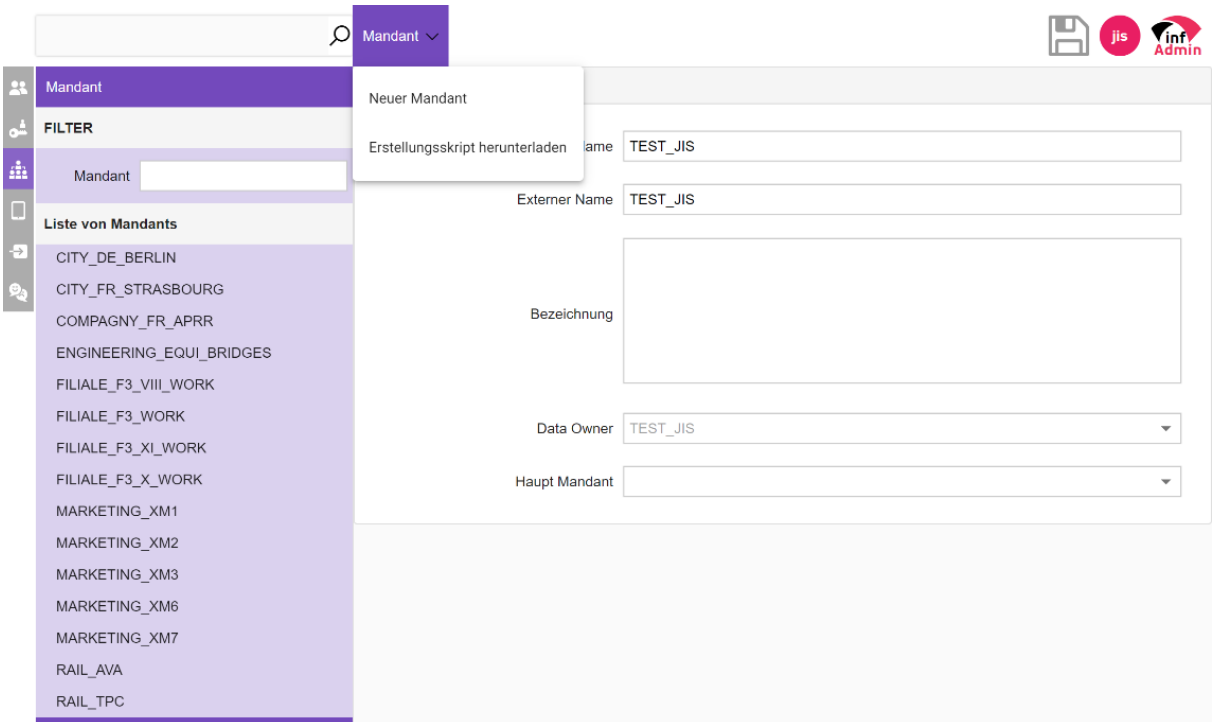
Pour réaliser ces nouveaux modes dans l'application, je crée donc 2 nouveaux composants. Chaque mode de l'application est présent sur la gauche de la page, est représenté par une icône et une couleur qui lui est propre et permet ainsi de naviguer entre ces différents modes en cliquant dessus.



E - Les différents modus dans InfAdmin

L'image de droite représente les différents modes présents dans l'application. Les deux modes entourés en rouge sont les modes que j'ai réalisés durant les développements. Les autres modes étaient déjà présents antérieurement à mes développements.

Pour chacun des deux modes, j'attribue une nouvelle couleur à mon composant pour qu'il soit facilement identifiable, j'intègre les composants classiques présents dans les autres modes tels que la barre de recherche sur

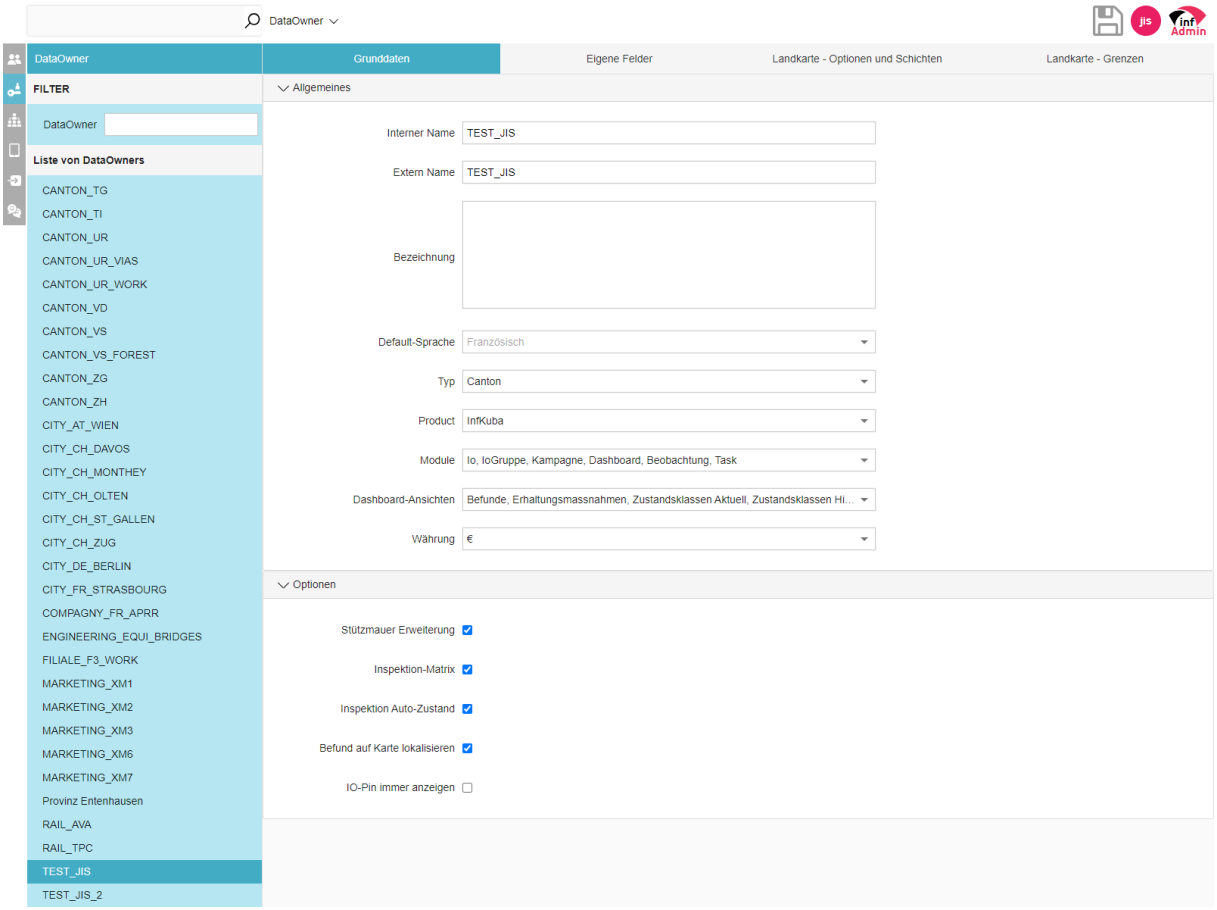


F - Onglet Mandant dans InfAdmin

la gauche et un onglet déroulant en haut de l'application. Enfin, j'intègre les fonctionnalités propres au mode.

L'image ci-dessus représente le mode « Mandant ». Ce mode permet de créer un mandant qui sera rattaché à un data-owner. Il n'a pas de propriétés avancées, uniquement des informations générales telles qu'un nom interne (administratif) et externe (public), un commentaire, le data-owner auquel il se rattache et s'il a un mandant parent.

Ce mode permet également de créer un nouveau mandant, ou de télécharger le script SQL qui permet de recréer ce mandant au travers d'un outil tel que PgAdmin.

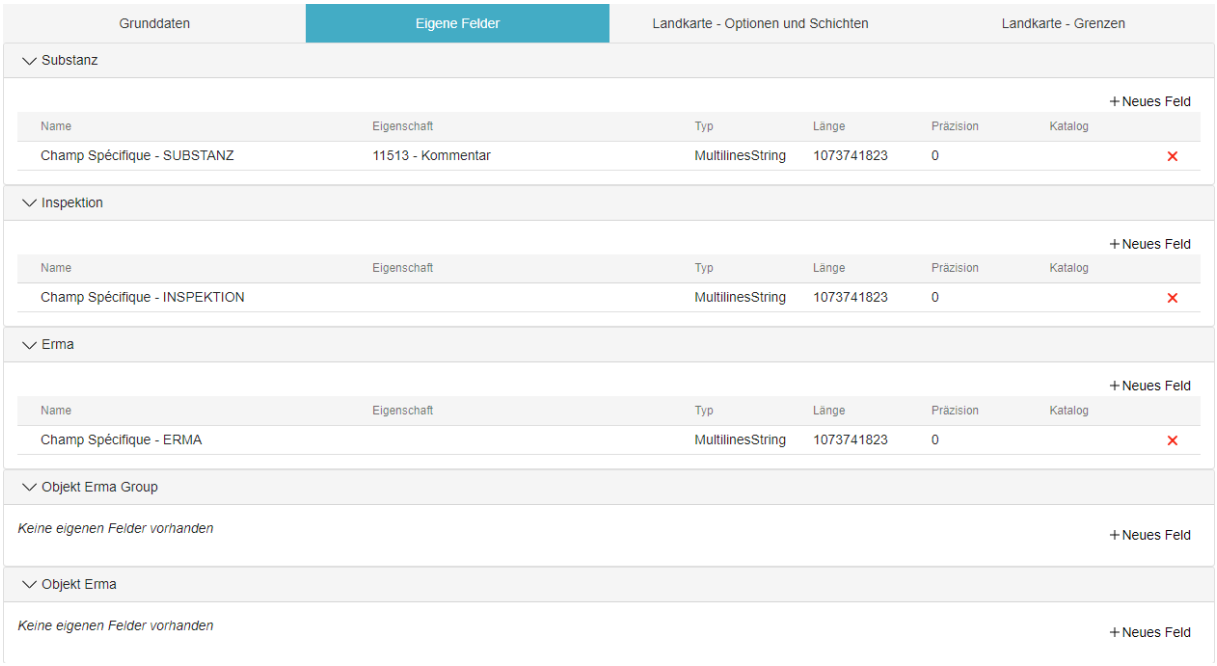


G - Onglet "Généralités" dans le mode DataOwner d'InfAdmin

Le second mode, représenté par l'image ci-dessus, est beaucoup plus complexe. Ce mode nécessite l'implémentation d'un autre composant : des onglets.

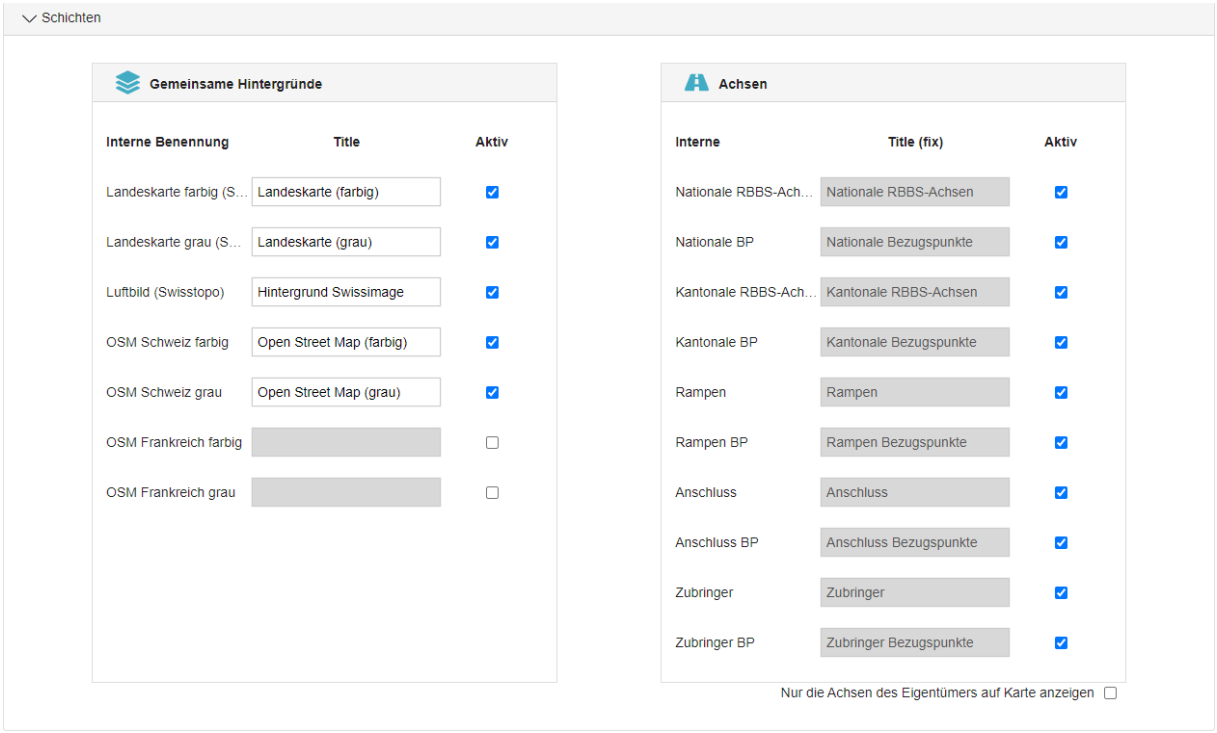
En effet, un data-owner est le point central de l'application. C'est à ce data-owner que toutes les données sont rattachées. Il possède des caractéristiques similaires aux mandants comme le nom interne, un nom externe, un commentaire

facultatif… Mais il possède également des configurations plus spécifiques comme le produit du data-owner (si c'est l'application InfKuba, InfVias, InfProtect…), la langue du data-owner, la monnaie et divers paramètres reliés au fonctionnement de l'application.



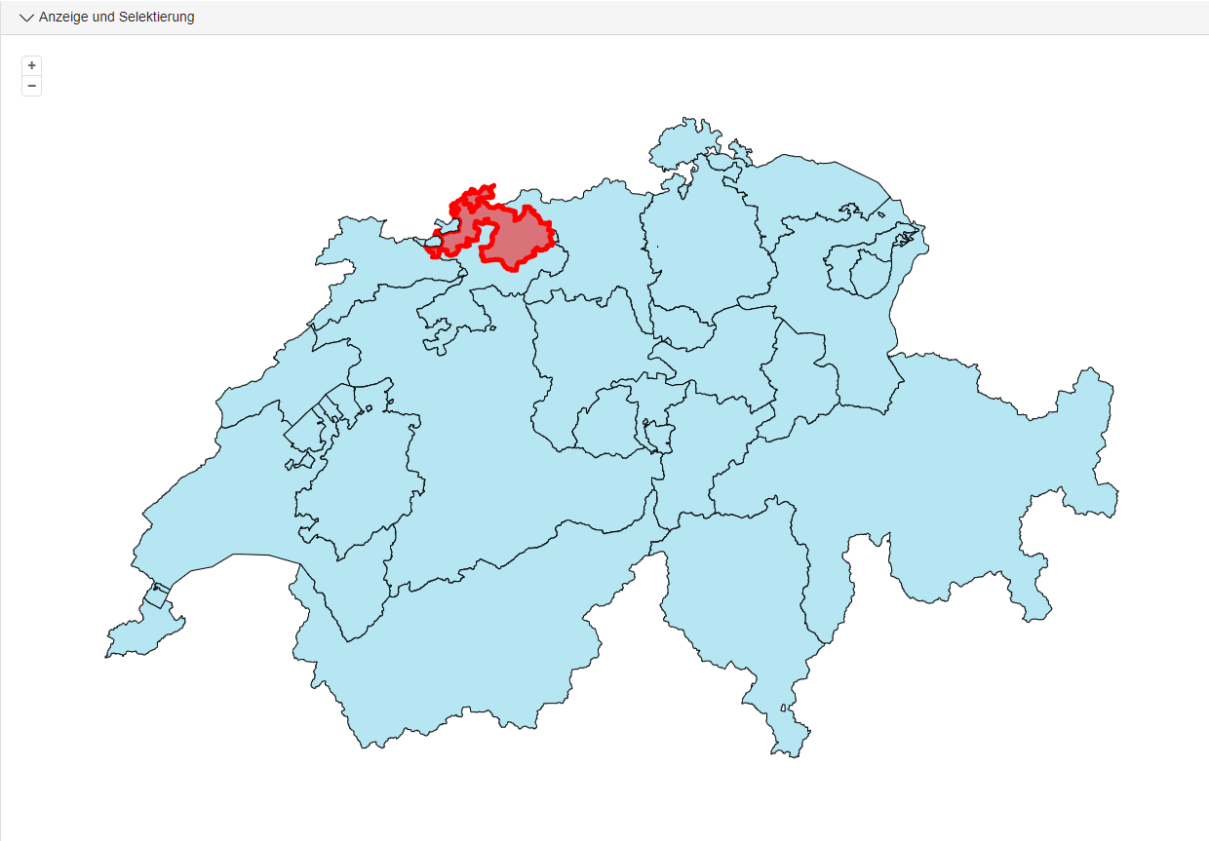
H - Onglet "Champs propres" dans le mode DataOwner d'InfAdmin

L'onglet suivant permet de configurer des champs propres. Un champ propre dans InfKuba, est un champ spécifique à ce data-owner, qui viendra s'ajouter au formulaire de saisie des informations de l'ouvrage. On peut par exemple attribuer par défaut un nom à l'ouvrage, mais le client peut par exemple décider d'ajouter un nouveau champ dans le formulaire qui correspond à la taille de l'ouvrage. Il est donc possible de configurer ce nouveau champ spécifique dans cet onglet.



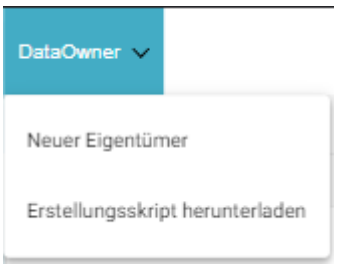
I - Onglet "Option et couches" dans le mode DataOwner d'InfAdmin

L'avant-dernier onglet permet de configurer les couches disponibles pour le data-owner. Les couches correspondent dans un premier temps aux fonds de cartes qui s'affichent sur la carte de l'application, mais il est aussi possible de configurer les axes qui seront visibles. Un axe correspond aux différentes infrastructures routières et est affiché de la même manière que les fonds de carte.



Enfin le dernier onglet « Landkarte – Optionen und Schichten », permet lui de configurer les zones géographiques allouées au data-owner. En effet, la Suisse par exemple, est découpée en cantons. Ces cantons ont des frontières théoriques qui permettent de délimiter la gestion administrative de chacun d'eux. Pour permettre d'accorder à un canton de créer des objets dans sa zone géographique, il faut donc lui configurer sa zone géographique de travail dans l'application. C'est dans cet onglet que tout est réalisé.

Tout comme pour les autres modes dans l'application, il est possible de créer un data-owner et de télécharger un fichier SQL permettant de le recréer, dans son état actuel, au travers d'un outil tel que PgAdmin. Le script SQL permet par exemple de restaurer des données potentiellement perdues depuis une autre base de données de backup ou de les importer dans une autre application.



J - Liste d'options supplémentaires dans le mode DataOwner

Comme j'ai pu le présenter précédemment, toutes les actions effectuées sur cette interface web passent au travers de services dédiés et d'une API pour faire la relation entre client et base de données. Je vais donc présenter dans la partie suivante la technique des développements que j'ai pu effectuer.

3. Réalisation technique

La partie précédente de ce document présente le client d'InfAdmin. C'est la partie graphique de l'application qui permet de rendre l'outil plus agréable à utiliser pour l'utilisateur.

Pour les développements, nous ne partons pas de rien. La structure de la base de données existe déjà et des données y sont déjà présentes. Je dois donc m'adapter à ce qui existe déjà, mais surtout comprendre comment sont structurées nos bases de données.

Comme je l'ai présenté précédemment, l'entreprise utilise des bases de données PostgreSQL pour InfKuba. Une base de données SQL permet de structurer les données et vise à réduire la duplication de données. Pour cela, les modèles de données sont fixes puisqu'ils doivent correspondre à la structure des tables mises en place.

L'onglet de gestion des mandants contient moins de fonctionnalités poussées, je me concentre donc sur la présentation de l'onglet de gestion des propriétaires de données.

En analysant donc les différents objets sur lesquels je vais travailler, je retrouve 3 objets principaux :

Les propriétaires de données (data-owners) sont les entités principales vers lesquelles pointent directement ou indirectement tous les autres objets de l'application. C'est dans cet objet que nous stockons les informations principales de l'objet source.

Les Eigene Felder[5] (EIFE) sont des champs spécifiques pour le data-owner. Il peut ajouter des informations supplémentaires pour les ouvrages, ou alors pour les inspections… Tous ces champs sont stockés dans la table « eatt_mkb » dédiée.

Et enfin, la dernière table concerne les configurations des zones géographiques accordées au data-owner. Chaque entrée dans cette table fait référence à une seconde table qui stocke réellement les couches sous leur forme brute. J'approfondirai le sujet un peu plus tard dans ce document.

Pour permettre de mettre à disposition toutes les informations du propriétaire des données, ou du mandant, il faut permettre à l'utilisateur d'initialiser les données.
Pour simplifier la création de chacun des objets, j'affiche un pop-up à l'utilisateur

[5] Traduction : Champs propres

qui va lui demander de renseigner les informations nécessaires à la création des objets. Tous les autres paramètres optionnels pourront être renseignés plus tard, donc je ne les affiche pas directement.

Lors de la création d'un data-owner, le service dédié dans l'application front-end va envoyer au travers d'une requête HTTP, toutes les informations renseignées par l'utilisateur. Chaque requête vers le serveur utilise une méthode bien spécifique au besoin évoqué.

Lors d'une requête HTTP, il faut utiliser la méthode adaptée qui sera utilisée pendant toute la durée du traitement de la requête. Dans mon cas, le service pour le Data-owner utilise 4 méthodes différentes : GET, POST, PUT, DELETE. D'après RFC Editor à l'origine du protocole HTTP, chacune des méthodes doit être utilisée comme suit :

- « La méthode GET demande le transfert [...] de la ressource cible ». Au travers de cette méthode, je demande au serveur de me retourner une ressource que lui seul possède.
- « La méthode POST demande le traitement d'une nouvelle ressource ». J'invoque cette méthode lorsque je veux créer une nouvelle ressource côté serveur.
- « La méthode PUT demande au serveur de modifier l'état actuel d'une ressource déjà existante ». C'est grâce à cette méthode que je peux demander au serveur de modifier un objet qui existe déjà en base de données. Cependant, « cette méthode est aussi utilisable dans le cas où la ressource serait potentiellement inexistante. » Si on cherche à modifier la ressource alors qu'elle n'existe pas, le serveur va alors créer la ressource demandée.
- « La méthode DELETE demande la suppression d'une ressource existante ».

Gestion du data-owner

Pour la création d'un data-owner, je préfère utiliser la double utilité de la méthode PUT du protocole HTTP. Lorsque je souhaite créer ou modifier mon entité, je fais appel à la même méthode et ainsi à la même URL. C'est le serveur qui se chargera de déterminer s'il faut le créer ou le modifier.

Le transfert des informations s'effectue au travers d'un objet JSON. Pour le propriétaire des données, l'objet ressemble à ceci :

```
{
    "productType": 1,
    "type": 1,
    "locale": "fr",
    "internName": "EXEMPLE_UHA",
    "externName": "Exemple pour l'UHA"
}
```

K - Exemple d'objet JSON envoyé au serveur pour la création d'un DataOwner

L'objet est très léger puisque c'est cet objet qui va être la source de toutes les autres relations avec les autres objets, il n'y a pas besoin de beaucoup d'informations pour l'initialiser. Tout le reste est géré côté serveur.

Côté serveur, j'ai pu implémenter les différents « endpoints » qui permettent de faire les fameuses requêtes HTTP. Un endpoint est une porte d'entrée sur le serveur qui permet d'effectuer diverses actions. L'API rest que j'ai pu développer à comme URL de base **https://infadmin.ch/api/**. C'est à partir de cette URL que je construis chacune de mes routes.

Chaque action correspond à une route, mais une route peut être utilisée pour plusieurs actions. Comme j'ai pu l'expliquer précédemment, lors de la requête http, je dois fournir la méthode utilisée. Ainsi sur une même route, je peux y implémenter plusieurs actions.

Par exemple pour récupérer un data-owner précis j'utilise la route : https://infadmin.ch/api/data-owner/ avec la méthode GET et en paramètre de la requête l'ID de l'entité. Cela me permet de récupérer un data-owner précis.

Sur la même URL, mais cette fois-ci avec une méthode PUT, je peux créer un Data-owner ou le modifier comme vu précédemment en lui envoyant avec la requête l'objet que je veux créer/modifier.

Mon nouvel objet est donc envoyé sur la route dédiée à la création et modification de gestionnaire de données, est récupéré par le serveur et me renvoi

l'objet correspondant à la nouvelle entité en base de données.


```
{
    "id": "3559d117-93ae-401f-b01c-b32582d3ee0a",
    "internName": "EXEMPLE_UHA",
    "externName": "Exemple pour l'UHA",
    "comment": null,
    "locale": "de",
    "type": 1,
    "productType": 1,
    "enabledModus": 7,
    "dashboardView": 0,
    "cultureId": "673ff05a-7e08-4095-8066-19d8ed0205df",
    "isSustainingEnabled": false,
    "isInspectionMatrixEnabled": false,
    "isInspectionAutoEnabled": false,
    "isBefundLocationEnabled": false,
    "alwaysShowIoPin": false
}
```


L - Exemple d'objet JSON renvoyé par le serveur du nouveau DataOwner

On remarque que le serveur a complété l'objet envoyé avec de nouvelles informations. C'est la représentation immédiate des informations stockées en BDD. Par exemple, l'objet a un ID, c'est avec cet ID unique que nous allons demander des informations ou des modifications au serveur.

Gestion des couches

Toutes ces informations sont le résultat d'un traitement du serveur qui a initialisé plusieurs données et configuration comme la configuration initiale de nos différentes couches.

Dans le concept initial, j'avais prévu d'initialiser les couches uniquement lorsque l'utilisateur cherche à accéder à la configuration d'une d'elles pour la première fois en cliquant sur les onglets dédiés de l'outil. Cependant, je me suis rendu compte lors de développement que cela pouvait provoquer un effet de bord en déclenchant une erreur dans InfKuba.

En effet, InfKuba n'initialise pas la configuration de la couche, mais ne fait que la lire. Ainsi, si la configuration est inexistante, l'application n'est pas en mesure de fournir les éléments d'affichage à OpenLayer et abandonne alors l'affichage.

J'ai donc modifié en cours de route, lorsque j'ai remarqué ce problème, le fonctionnement de l'outil.

Le nouvel objectif est donc d'initialiser toutes les configurations minimales requises dès le départ, de rendre l'application utilisable très rapidement et pour

respecter le cahier des charges, de permettre, même à une personne non qualifiée, d'initialiser les données requises.

Les couches GIS sont fournies par les différents serveurs GIS dédiés. Ces couches GIS permettent d'obtenir un fond de carte dans l'application InfKuba. Pour obtenir ces fonds de cartes, il faut renseigner l'URL de base pour requêter le bon serveur, puis à l'aide d'OpenLayer, de multiples requêtes sont faites pour obtenir des tuiles d'images qui seront juxtaposée dans le bon ordre pour former un fond de carte.

Toutes les couches utilisables dans l'application sont stockées dans une table dédiée (« gis_layer ») contenant les URL pour contacter les serveurs GIS et d'autres informations complémentaires non-essentielles dans mon cas. Quand l'utilisateur veut sélectionner un fond de carte, ce n'est pas directement dans cette table que l'application va aller chercher.

Pour permettre de personnaliser les couches d'un data-owner, des entités différentes sont créées dans une autre table. Ci-dessous une représentation de la table « dao_gislayer_ref » qui permet de créer des couches personnalisées :

gao_id	Id de l'entité
gao_dao_id	Id de l'entité 'data-owner' attaché
gao_title_vl	Titre personnalisé de la couche
gao_opacity_vl	Opacité de la couche
gao_gly_id	Id de la couche GIS attachée
…	…

Chaque entrée dans cette table correspond à une couche active pour le data-owner. Elle contient l'ID faisant référence à la couche GIS originelle dans la première table et d'autres informations de personnalisation tel que le nom qui sera affiché, l'opacité…

Quand le data-owner est initialisé, je lui crée donc des couches actives par défaut pour permettre de le rendre utilisable sans avoir besoin de le configurer et éviter des effets de bords comme j'ai pu le décrire plus tôt.

Pour cette première phase de développement, il n'est possible d'activer ou de désactiver que certaines couches prédéfinies. Je prends uniquement les couches générales telles qu'Open Street Map, Carte nationale et Swisstopo. Lorsque l'utilisateur rend une couche active, je crée une nouvelle entrée dans la base de données avec les informations fournies. S'il ne définit pas de nom pour cette couche, un nom par défaut lui sera attribué en fonction de la langue du data-

owner. Lorsque l'utilisateur désactive une couche, alors je supprime l'entrée dans la base de données.

Pour la création de couches pour les axes, le fonctionnement est similaire, ce ne sont que les couches et les données propres à ce type de couches qui varient. Toute cette configuration se fait dans l'interface, que j'ai implémenté pour ce besoin que j'ai pu présenter plus tôt dans l'onglet « Landkarte – Optionen und Schichten ».

Maintenant que mes couches sont initialisées, il faut définir les zones auxquelles le data-owner est associé dans l'application, pour permettre d'y créer de nouvelles données.

Gestion des bordures

Tout comme pour l'initialisation de couches, j'avais prévu de faire cette première initialisation lors de la première requête pour récupérer les bordures du data-owner. Cependant, pour éviter des problèmes dans l'application InfKuba, cette initialisation se fait au même moment que pour le data-owner.

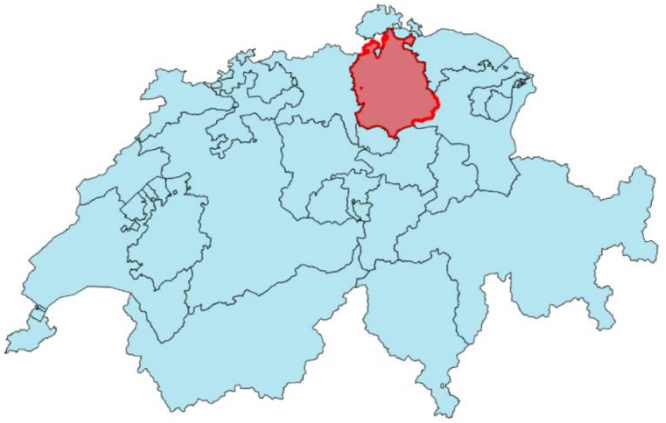
Mettre en place les couches disponibles pour l'utilisateur permet de générer des fonds de carte et une interface avec des pré-données pour que l'utilisateur puisse saisir de nouvelles informations.

Il faut cependant s'assurer qu'il ne puisse pas saisir des informations en dehors de sa zone de travail. Autant la saisie de ces informations sur l'application en dehors de sa zone de travail ne poserait pas de gros problèmes en dehors de données incohérentes, autant cela pourrait devenir un problème lors de l'export de données vers d'autres applications qui peuvent avoir des requis plus stricts.

Les zones de travail sont délimitées par des bordures que je peux récupérer auprès d'un serveur GIS. Lorsque je veux permettre à l'utilisateur de définir sa zone de travail, je récupère les bordures et avec la librairie OpenLayer, je les affiche sur une carte interactive.

Pour ne pas dépendre d'un service supplémentaire, il a été choisi de télécharger les fichiers contenant les différentes bordures et de laisser le serveur API d'InfAdmin délivrer ces couches d'informations. Ci-dessous, un exemple du fichier contenant les données brutes (à gauche) et leur correspondance graphique une fois traitées (à droite).


{"type":"FeatureCollection","features":[{"type":"Feature","id":"CANTON_GRENZEN.1","geometry":
{"type":"MultiPolygon","coordinates":[[[[2695054.51823642,1159488.07568353,2765.95375],[2695281.
48704777,1160449.89325988,2464.535],[2695169.19625059,1160949.48267064,2458.57875],[2695169.50181772,
1161172.9758891,2423.52875],[2694908.29980019,1161527.58307543,2704.30375],[2694904.29266269,1161559.
75998816,2702.61375],[2694863.88687325,1161640.86148136,2699.99625],[2694906.01436899,1161722.
78014435,2709.1375],[2694988.9175227,1161747.78295552,2715.2925],[2694984.00139756,1161740.15039336,
2717.01375],[2694902.87334822,1161791.11710673,2724.00625],[2694983.44667717,1161823.28112500,2711.
21625],[2694896.78422593,1161836.65369662,2734.98375],[2694887.50669871,1161851.37516045,2735.09875],
[2694985.55926963,1161873.34826222,2745.17875],[2694893.47179191,1161894.88847175,2749.30125],
[2694922.08148571,1161933.73409545,2725.2525],[2694927.92280692,1161976.58868441,2715.99],[2694905.
04757887,1162006.46314999,2702.02875],[2694930.07200059,1162026.64956892,2700.8625],[2694943.
14554855,1162060.32767663,2707.86125],[2694933.05853714,1162077.01413696,2715.87125],[2694985.
48257358,1162168.9040406,2618.76],[2695026.97015728,1162201.30896246,2640.09125],[2695055.74490641,
1162242.23451484,2580.64125],[2695066.86112773,1162272.11661613,2584.89875],[2695074.41879979,
1162316.11355127,2587.09125],[2695076.09587994,1162346.47129506,2597.63875],[2695054.56682752,
1162421.16435414,2634.5975],[2695032.21708711,1162526.12462159,2700.13],[2695041.55277907,1162543.
75520097,2687.7275],[2694998.24097952,1162615.96029745,2712.1925],[2695001.42888846,1162631.90753127,
2713.59375],[2694984.91904598,1162678.87785409,2730.79],[2694981.05067855,1162707.99961766,2738.
9575],[2694925.2095018,1162760.2410332,2761.425],[2694858.14112171,1163333.34574149,2486.57125],
[2694989.8438418,1163408.54889901,2512.8825],[2694824.4711325,1163937.84032115,2706.1775],[2694778.
4135685,1163920.60622727,2700.46875],[2694730.12633743,1163908.3120693,2686.96125],[2694720.9174111,
1163919.59260429,2686.02125],[2694698.89189713,1163919.45392473,2684.08625],[2694681.66775237,
1163921.35406835,2683.83375],[2694678.39270675,1163929.96570927,2683.475],[2694646.26819213,1163925.
35436539,2687.3125],[2694621.78768281,1163941.57890484,2691.2275],[2694571.1111831,1163928.16429383,
2701.57625],[2694534.5149944,1163912.98836643,2704.07875],[2694416.32998833,1163891.57326946,2715.
02375],[2694143.32237531,1163941.58077375,2832.625],[2694089.96967799,1163966.12698002,2831.34375],
[2694036.99987271,1164004.06416634,2838.3975],[2693947.02142476,1164058.13216652,2857.225],[2693871.
05515427,1164099.75347726,2878.5575],[2693852.28300346,1164113.03342224,2890.01375],[2693836.
33413473,1164182.33678322,2917.30625],[2693826.08201886,1164203.49382941,2924.325],[2693825.73994306,
1164232.871367,2906.61],[2693838.87291966,1164273.58977416,2889.06625],[2693767.80264058,1164300.
83608986,2857.4275],[2693737.07428891,1164305.58117386,2838.6775],[2693703.46387168,1164345.59364833,
2831.8575],[2693660.4787842,1164380.43034277,2789.635],[2693641.51664211,1164389.26174172,2779.7125],
[2693613.72265727,1164423.59837382,2773.70375],[2693571.12306431,1164426.65483644,2760.73875],



M - Représentation BRUTE et RENDERED d'un fichier shape

Lorsque l'utilisateur sélectionne une nouvelle zone de travail, son ID est ajouté dans la liste des différentes zones actives pour le data-owner et est colorée en rouge sur la carte interactive.

Une zone active correspond à une entrée dans la base de données. Cela permet de gérer les configurations des bordures au cas par cas même si elles sont attribuées au même data-owner.

Lorsque InfKuba se lance, elle va faire une requête au serveur GIS avec en paramètres les ID des différentes bordures actives qui correspondent aux zones de travail. C'est cette zone de travail qui va limiter l'utilisateur dans la saisie des informations dans le data-owner.

Gestion des champs propres

La gestion des champs propres est une autre fonctionnalité que j'ai pu implémenter dans l'application. Comme je l'ai expliqué plus tôt, les champs propres permettent d'ajouter des renseignements aux différents ouvrages, propres aux besoins du data-owner.

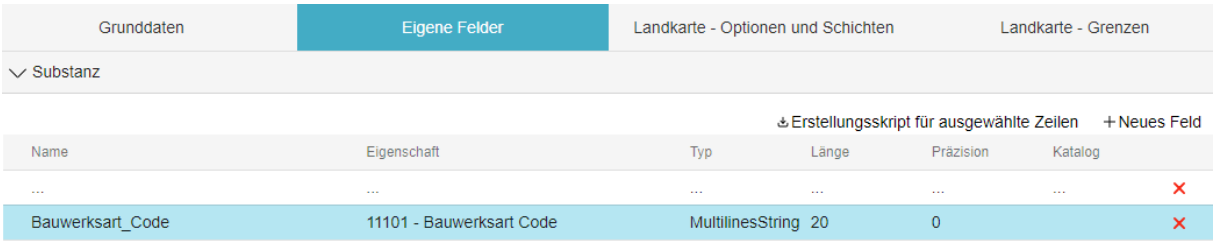
Les champs propres se retrouvent à plusieurs endroits dans l'application. Dans les informations globales de l'ouvrage, dans les rapports de constatation et dans les rapports d'intervention.

Je prends l'exemple d'un client dans l'application InfKuba. Il a besoin de nouveaux champs spécifiques à ses besoins qui ne se retrouveront que dans son environnement. Dans les informations de l'ouvrage où retrouve une catégorie « Allgemeine eigene Felder [6] ». Ce sont ces champs qui sont spécifiques à un data-owner et qu'on ne retrouve chez aucun autre client.

[6] Traduction : Généralité champs propres

Ici, il décide de rajouter différentes informations qui doivent permettre une gestion interne à l'office cantonale concernant ses objets d'infrastructure.

Je prends l'exemple d'un data-owner et vais chercher dans l'onglet des champs propres un exemple que nous pouvons retrouver dans l'application.

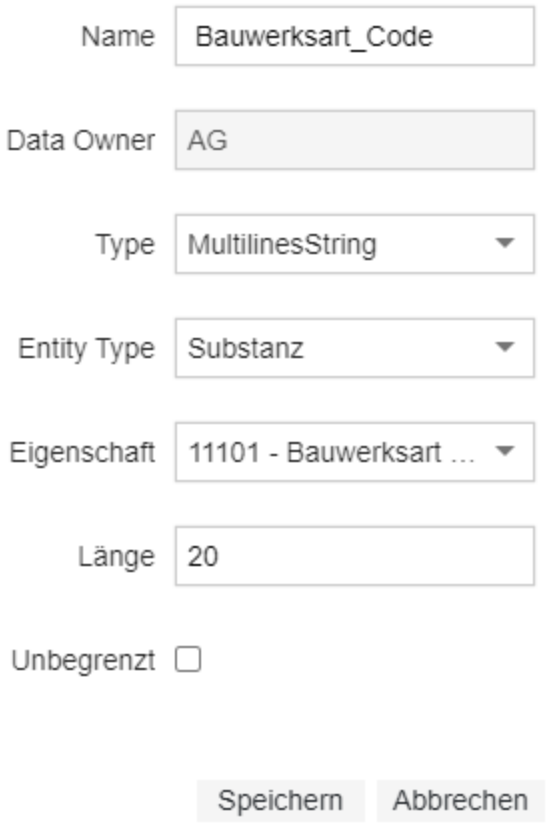


N - Onglet EIFE de note DataOwner modus

Cette ligne correspond à un champ dédié à un client qui est stocké en base de données pour le data-owner. Le tableau résume les propriétés du nouveau champ et permet de le supprimer si besoin.

Quand un utilisateur crée un champ propre, il devra fournir différentes informations pour permettre de personnaliser ce nouveau champ. Je crée donc une fenêtre séparée dans laquelle l'utilisateur peut renseigner les différentes propriétés souhaitées de ce champ.

Certaines propriétés sont relatives au type de champ souhaité. Par exemple, les champs « texte » ou « texte multiligne » peuvent avoir une longueur maximum ou alors être « infini » en cochant la case dédiée. Si le champ est du type décimal, il peut aussi avoir une taille maximum et une précision (nombre de chiffres après la virgule). D'autres types de champs existent et possèdent des propriétés spécifiques à leur type.



O - Modale de création et modification d'un EIFE

L'objet Eigenschaft [7] est une autre entité, qui n'est pas relative à mon data-owner, mais à l'application InfKuba. Je dois donc chercher en base de données toutes les propriétés qui existent en filtrant

[7] Traduction : Propriété

uniquement celles qui correspondent à mon type d'entité (pour rappel, le type d'entité permet de définir où apparaîtra le nouveau champ dans l'application.).

Quand l'utilisateur est satisfait des changements, il peut sauvegarder l'objet en envoyant une requête au serveur pour lui demander de le stocker en base de données. Côté serveur, je récupère l'objet généré, je fais une liaison avec le data-owner, je fais une relation avec l'Eigenschaft, puis je fais persister les informations en bases de données tout comme je le ferais pour les zones de travail ou les informations basiques du data-owner.

Export des scripts SQL

Une autre fonctionnalité attendue décrite dans le cahier des charges, est le fait de pouvoir exporter la configuration du data-owner sous la forme de requêtes SQL brutes. L'avantage, est que nous pouvons restaurer nos données très simplement en exécutant une requête déjà préconstruite contenant les anciennes données, sans avoir besoin de recréer l'objet étapes par étapes.

Je rajoute donc un nouveau bouton dans l'entête de la page qui permet de télécharger cette configuration. Pour générer un tel fichier je dois faire une requête spécifique à la base de données. J'utilises la possibilité qu'offre EntityFramework d'exécuter des requêtes dites « Raw », c'est-à-dire des requêtes qui sont écrites à la main par le développeur et non pas une requête qui est générés par l'ORM au travers de méthodes.

J'appelle donc une méthode en base de données qui s'appelle « F_GENERATE_INSERT_SCRIPT », qui va, en lui fournissant le nom d'une table et l'ID des entités dans cette table qu'on souhaite exporter, nous retourner la requête SQL de type « insert » permettant de recréer nos lignes. Je n'ai pas créé cette méthode en base de données, elle a été créée par un autre développeur, je ne fais que l'utiliser, je ne m'attarderais donc pas à l'explication de son fonctionnement.

J'invoque cette méthode pour les tables « dataowner_mkb », « dao_gislayer_ref » et « dao_shape » en fournissant l'ID du data-owner. Une fois le traitement par la base de données terminé, je récupère les différentes requêtes générées et les regroupe dans un même fichier que je renvoie au client. Lorsque le client reçoit le fichier généré, il le télécharge automatiquement sur l'ordinateur de l'utilisateur en mettant le nom du data-owner comme nom de fichier.

Une fois les développements effectués, je souhaite m'assurer que mon application fonctionne correctement en réalisant des tests automatiques.

Tests automatiques

De manière générale, une bonne pratique est d'écrire des tests automatiques **avant** de commencer les développements de l'application. Cela permet de s'assurer que les développements respectent le cahier des charges et que les nouveaux développements ne cassent pas d'autres fonctionnalités de l'application.

Cependant, il peut arriver que les tests ne soient pas réalisés au début des développements, mais pendant ou après les développements. C'est le cas ici. J'ai réalisé les développements des tests de mes différentes fonctionnalités à la fin des développements.

En effet, lorsque j'ai été chargé des développements, j'avais une unique année d'expérience au sein de l'entreprise. L'outil d'administration visant à modifier les objets métiers de l'application InfKuba, je n'avais pas encore assez de connaissances sur l'architecture et le fonctionnement de l'application pour savoir ce que j'attendais exactement.

Par exemple, quand j'ai commencé l'initialisation du data-owner, je me suis rendu compte bien plus tard qu'il fallait initialiser les couches et les bordures en même temps pour éviter un effet de bord. C'est précisément ce genre de comportement que l'on souhaite vérifier avec des tests automatiques.

Ayant terminé l'implémentation de mes fonctionnalités, je commence la création de différents tests côté serveur pour m'assurer que je respecte le cahier des charges et que le serveur fonctionne comme attendu.

L'écriture d'un test est similaire à des développements standards. La seule différence, c'est que je n'attends pas des données provenant d'utilisateurs, mais je fournis les données d'entrée pour savoir exactement ce que le serveur devrait me retourner.

Pour réaliser mes tests, j'utilise le framework Xunit qui a été développé spécifiquement pour .NET Framework. J'ajoute à chaque méthode de test le mot-clef Fact, pour permettre à Xunit de savoir que c'est une méthode de test à exécuter. Dans cette méthode de test, j'appelle les méthodes dont je souhaite vérifier le comportement, puis je compare le résultat retourné par ma méthode avec le résultat attendu.

L'exemple ci-dessous nous permet de tester la création d'un data-owner.

```
[Owner("JIS")]
[Fact]
public void InitializeDataOwner_DefaultModulesAreSet()
{
    DataOwner dataOwner = new();
    _dataOwnerMediator.InitializeDataOwner(dataOwner);

    dataOwner.EnabledModus.Should().Be(EnabledModus.Io | EnabledModus.IoGruppe
| EnabledModus.Kampagne);
}
```

P - Exemple de code d'un test unitaire.

Dans ce test, j'initialise un nouveau data-owner à partir d'un modèle vide, comme si l'utilisateur ne fournissait aucune donnée et je vérifie que les valeurs par défaut ont été initialisées. Dans mon exemple, je teste l'initialisation des modules par défaut, pour éviter les conflits avec la base de données qui requiert des valeurs non nulles.

Je peux comparer un data-owner modèle avec le résultat de la méthode ou alors tester les valeurs individuellement comme ce que je fais ici.

```
[Owner("JIS")]
[Fact]
public void InitializeDataOwner_LayerAndLayerAxisCalled()
{
    _dataOwnerMediator.InitializeDataOwner(new DataOwner());

    _gisLayerMediator.Verify(glm => glm.InitializeLayers(), Times.Once);
    _gisBorderMediator.Verify(glm => glm.InitializeLayers(), Times.Once);
}
```

Q - Exemple de code d'un test unitaire

Le test ci-dessus que j'ai implémenté me permet de vérifier que lors de l'initialisation de mon data-owner, j'appelle les méthodes d'initialisation de mes couches et de mes bordures. Je ne teste pas le **résultat** des méthodes qui me retournent les nouvelles entités, mais uniquement **si elles sont appelées**. Le résultat des méthodes en lui-même est testé dans des fichiers dédiés aux couches et aux bordures.

Ces tests me permettent de garantir que l'intégration de mon développement colle avec les attendus et que je ne risque pas de générer d'erreur inattendue lors de la mise en production à cause d'une méthode défaillante.

Une fois mes tests rédigés, je les exécute en local sur ma machine pour m'assurer que tout est correct et j'applique les correctifs si nécessaire. Une fois que tous les tests sont validés, je peux demander à mettre mon travail sur la branche de développement au travers d'une demande de fusion sur l'outil Devops.

Avant de valider la fusion de mon travail avec la branche de développement, le serveur exécute automatiquement les tests sur un autre environnement que l'environnement local du développeur, pour s'assurer qu'il n'y a pas d'erreur même sur une configuration de poste différente.

Si les tests ne sont pas validés, le développeur doit d'abord appliquer les correctifs requis puis les tests seront à nouveau exécutés avec les nouveaux changements. À cette étape, si tout est validé, alors le travail est mis à disposition sur la branche développement accessible par tous les développeurs.

C'est à ce moment que les développements sont considérés comme terminé et je peux demander la validation de ma tâche en faisant passer son état sur Jira de « en cours » à « Analyse de la qualité de code ».

Rendu et finalité

L'étape de validation de la qualité de code se retrouve en dehors de l'implémentation de changements. Cette étape est un test réalisé manuellement par une personne de l'équipe dédiée à tester les nouvelles fonctionnalités.

Cette personne va tester que l'application répond au cahier des charge de base, même si les tests automatiques sont censés reprendre le cahier des charges au global, il peut arriver que certaines parties soient manquantes. Cette phase de tests essaye de rendre presque nul le risque de retrouver ce problème en production.

Cette étape manuelle de tests permet également de s'assurer de l'expérience utilisateur fournie par les développements. Cela permet de s'assurer qu'il n'y a pas d'erreur de traduction, que les différents éléments soient à la bonne place, qui ai les bonnes interactions…

S'il manque quelque chose ou que des erreurs ont réussies à se glisser dans le rendu, une nouvelle demande de changements sera assignée au développeur, en détaillant le problème et ce qui est attendu.

Une fois que j'ai validé toutes ces étapes, la fonctionnalité que j'ai créée est déclarée utilisable et apte à être mise en production. Ma fonctionnalité a été donc été rajoutée au déploiement de la nouvelle version qui comprend d'autres

améliorations et est actuellement utilisable au sein de l'entreprise pour manipuler les données dans l'entreprise.

L'application a déjà été utilisée plusieurs fois pour initialiser les données de nouveaux clients et est aussi utilisée pour la modification des informations de différents clients si besoin.

L'outil a déjà par exemple été utilisé dans le cadre de la création d'un data-owner de démonstration pour faire une démo de fonctionnement à de potentiels nouveaux clients.

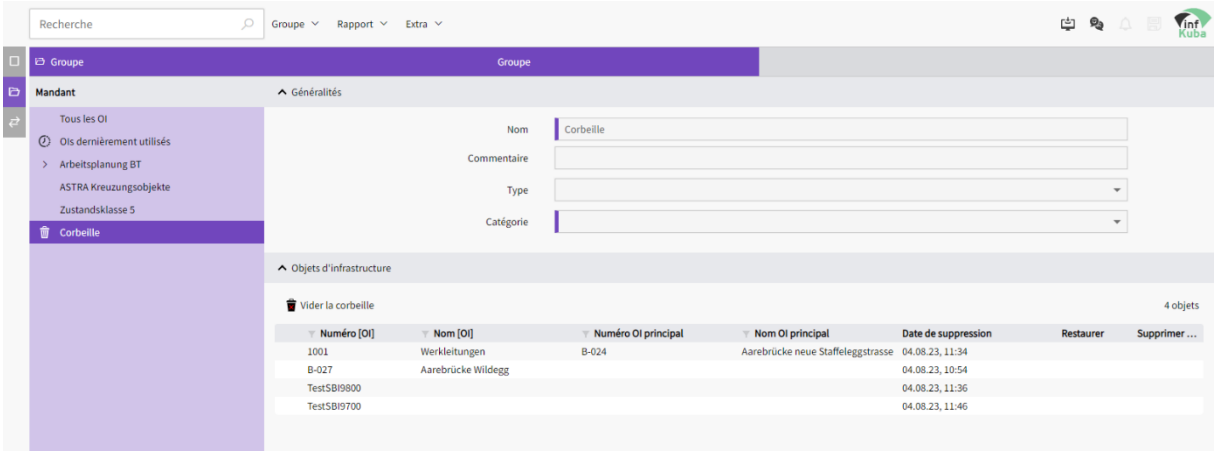
L'application InfKuba continue de générer des besoins différents au fur et à mesure de son évolution et elle a, depuis la mise en production des développements détaillés dans ce rapport, d'ores et déjà requis l'implémentation de nouveaux développements pour la gestion des data-owner.

4. Au-delà d'InfAdmin

Les développements que j'ai pu effectuer sur InfAdmin mon permis de mieux comprendre le cœur du fonctionnement d'InfKuba et de m'ouvrir de nouvelles opportunités de développements sur l'application mère.

J'ai notamment pu contribuer à maintenir l'application à jour en mettant à niveau les différents modules utilisés dans l'application vers leurs nouvelles versions et ne pas être dépassé par les mises à jour. Ce genre de mise à jour est nécessaire autant pour le côté fonctionnel, que pour le côté sécurité.

J'ai également réalisé dans l'application InfKuba, les développements d'une nouvelle fonctionnalité permettant d'envoyer des objets d'infrastructure dans une corbeille, plutôt que de les supprimer définitivement immédiatement.



R - Nouveau groupe d'objet statique dans InfKuba

Cette nouvelle fonctionnalité se base sur des éléments déjà existants que j'ai dû modifier en m'assurant de ne pas altérer le fonctionnement antérieur à mes développements de ces objets.

Cette modification a requis des modifications sur le côté client de l'application, mais également sur le côté serveur et ainsi la réalisation de tests pour les développements.

Enfin, une dernière contribution que j'ai pu apporter concerne l'outil de conversion que j'avais réalisé l'année dernière lors de ma première année au sein de l'entreprise. Pour rappel, cet outil de conversion est une application CLI qui vise à convertir des coordonnées brutes, en objets stockés dans les bases de données d'InfKuba.

Elle va pour cela chercher à positionner l'objet dans son contexte et recherche l'élément stocké en base de données correspondant le mieux à l'entrée brute.

Les changements que j'ai pu effectuer concernaient la mise à jour des données que l'utilisateur peut fournir. En effet, cet outil était à destination d'un client bien spécifique et ainsi, son utilisation de l'outil lui a généré un nouveau besoin. Ce nouveau besoin inclut de nouvelles données non traitées par l'application, mais nécessitait cependant un traitement.

J'ai donc récupéré le cahier des charges concernant l'outil qui avait été mis à jour avec ce nouveau besoin et ai réalisé les développements. Une fois les développements réalisés et validés par le workflow, le client à obtenu la nouvelle version de son outil contenant le traitement des nouveaux objets.

Les différentes réalisations sur les multiples projets portés par Unit Solutions m'ont mené vers la fin de mon année scolaire et plus particulièrement de ma première année de master avec l'écriture de ce rapport.

Conclusion

Je suis pleinement satisfait de cette première année de master que j'ai pu effectuer chez Unit Solutions.

Travailler sur des projets en relation directe avec le cœur de l'application InfKuba m'a apporté de nouvelles connaissances aussi bien théoriques que pratiques sur la manière de développer ou de gérer les développements d'une application d'une telle envergure.

La possibilité de travailler autant sur l'application mère, sur l'outil d'administration et sur des outils externes en parallèle m'a permis de voir plusieurs facettes de l'application pour comprendre comment tous les éléments entrent en corrélation les uns avec les autres.

Les développements effectués sur l'outil d'administration sont maintenant disponibles en production et utilisés pour créer toutes les nouvelles entités nécessaires pour les nouveaux clients ou les clients déjà existants. Elle permet d'économiser beaucoup de temps et de flexibilité sur l'administration de l'application.

Mon tuteur m'a accordé une pleine confiance en m'attribuant des tâches complexes à fort intérêt pour l'entreprise et a également été présent tout du long pour répondre à mes questions et m'accompagner pour mieux comprendre le contexte et tout l'environnement autour d'InfSuite.

J'ai hâte de reprendre de nouveaux développements pour Unit Solutions lors de ma seconde et dernière année en master.

« Julien a rejoint notre équipe de développement depuis 2 ans.

Lors de cette deuxième année, il a continué à progresser au niveau technique (.NET, Angular) en travaillant principalement sur l'extension de l'outil interne d'administration Web de la suite logicielle « infSuite » ainsi que sur des outils d'initialisation de données géographiques.

En fournissant un travail de qualité sur des sujets de plus en plus pointus, Julien a démontré qu'il pouvait prendre part à des tâches plus importantes au sein de l'équipe : il participe depuis quelques mois aux revues de code des autres développeurs seniors, prête main-forte lors des périodes de stabilisation avant les releases principales, et en cette fin de 2ème année commence ses premiers

développements de Change Request sur les modules externes de la suite logicielle « infSuite ».

Nous nous réjouissons d'avoir Julien pour sa troisième année où il ne manquera pas d'apporter sa bonne humeur et son expérience dans l'équipe. Cette dernière année dans le projet devrait être l'occasion pour lui de progresser en rapidité de développement afin de devenir un très bon développeur junior. »

M. Cédric Martin, chef de projet de la suite logicielle « InfSuite ».

Glossaire :

Terme	Définition
Framework	Un Framework est un ensemble des composants autonomes qui permettent de faciliter le développement d'un site web ou d'une application.
API	Une interface logicielle qui permet de « connecter » un logiciel ou un service à un autre logiciel ou service afin d'échanger des données et des fonctionnalités.
Back-End	C'est le serveur d'une application qui est invisible à l'utilisateur mais qui permet de faire le lien entre la base de données et le monde extérieur de façon sécurisée.
Front-End	C'est la partie cliente qui s'affiche pour l'utilisateur et qui permet d'interagir avec les données de l'application.
JSON	Le JSON est un format textuel de représentation de données très utilisé en informatique.
CLI	Une application Command Line Interface est une application qui s'exécute dans la console du système d'exploitation.
OFROU	L'Office Fédérale des Routes est l'autorité suisse compétente pour les infrastructures routières.

Annexes :

Résumé

Ce document présente mon expérience au cours de ma première année en tant qu'étudiant en master au sein de l'entreprise Unit Solutions. Il met en évidence ma participation significative à l'un des projets majeurs de l'entreprise, au cours duquel j'ai conçu un outil d'administration au cours des 9 mois de mon immersion professionnelle. Ce rapport détaille l'acquisition de compétences durant cette période et aborde les divers défis auxquels j'ai été confronté, ainsi que les solutions que j'ai pu apporter pour les résoudre.

Mots-Clefs :

- **Outil d’administration**
- **Base de données**
- **Développement objet**
- **Méthode Agile**